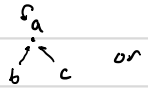
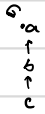
Can represent a set w/ a tree

{a}

{a,b}

{a,b,c}





$\pi(a)=a$ ∴ a is root

$\pi(b)=a$

$\pi(c)=b$

rank(c) = 0
rank(b) = 1
rank(a) = 2

Def'n: $\pi(x)$ = parent node (think "pointer")

root node = the x s.t. $\pi(x)=x$ ← use the root node to ID sets

rank(x) = # edges in longest simple path from x to a leaf node

∴ def make_set(x)

$\pi(x)=x$ Set parent to self O(1)

rank(x) = 0

~~def find(x)~~

~~while $x \neq \pi(x)$~~

~~$x = \pi(x)$~~

~~return x~~

look for root node

#ops = depth
3 in this case

def find(x) ← Path compression

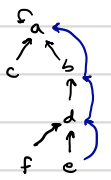
while $x \neq \pi(x)$

$\pi(x)$ = find($\pi(x)$)

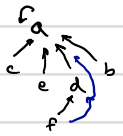
return $\pi(x)$

Two pass method:
1. up to find root
2. make each node on found path point to root node

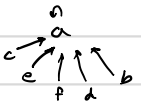
eg



find(e)



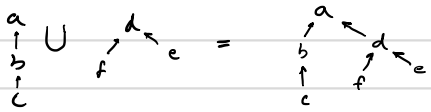
find(f)



Subsequent find() on this tree O(1)

requires amortized analysis
i.e. over a sequence of operations

But what about union?



Note: attach smaller ranked tree to larger one so find() won't take too long

thus ← aka, union by rank

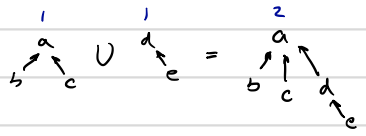
```
def union(x,y)
    r_x = find(x)
    r_y = find(y)
    If r_x = r_y
    |   return
    If rank(r_x) > rank(r_y)
    |   π(r_y) = r_x
    Else
    |   π(r_x) = r_y
    |   If rank(r_x) = rank(r_y)
    |   |   rank(r_y) += 1
```

Kruskal running time w/ tree data structure

```
def Kruskal_MST(G, w)
    A = ∅
    for all v ∈ G.V                          ┐
    |   make_set(v)                         ┘ O(|V|)
    G.E = G.E sorted in non-decreasing order by w  → O(E log E) = O(E log V)
    for all (u,v) ∈ G.E                      ┐
    |   if find_set(u) ≠ find_set(v)         |
    |   |   A = A ∪ {(u,v)}                  | 2nd loop
    |   |   union(u,v)                       |
    return A                                 ┘
```

since $E \le V^2$
$\Rightarrow \log E \le 2 \log V$
$\Rightarrow E \log E \le 2E \log V$

Observe: 1. $\forall x$, rank(x) < rank(π(x))



2. Root node w/ rank K is formed from the union of 2 rank K-1 trees
3. Any root node of rank K has at least 2^K nodes in the tree

Proof: Induction K = 0 & $2^0 = 1$ ✓

Assume true @ K-1. By prop 1, need to union 2 trees of rank K-1, each has 2^{K-1} nodes
$\Rightarrow$ tree w/ root of rank K has $2^{K-1} + 2^{K-1} = 2^K$ nodes □

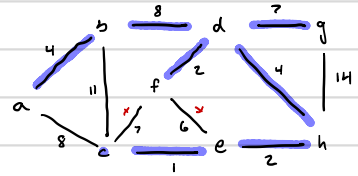
4. With |V| nodes, max rank is $\log_2 |V|$

∴ find(·) = O(log|V|)
union(·) = O(log|V|) ← 2 finds + constant work ∴ 2nd loop O(|E| log|V|)

So total running time O(|E| log|V|)

Prim's algorithm

Similar, but iteratively grows tree
Form cut w/ nodes used so far, pick lightest edge crossing that cut



See text for more details

Huffman encoding

Used in MP3 compression

Data: string S (very long) over an alphabet Γ (usually $|\Gamma| >> 2$)

Goal: Find a binary encoding of Γ resulting in a minimal encoding length S_e

eg $\Gamma = \{a, b, c\}$ Statistics of S:

	a	b	c
# char	45	16	2

fixed-length encoding: $a \to 00$, $b \to 01$, $c \to 10$ $\quad |S_e| = 2 \cdot 45 + 2 \cdot 16 + 2 \cdot 2 = 126$

Wasted space... what about variable length encoding?

$a \to 0$, $b \to 10$, $c \to 11$ $\quad T_e = 1 \cdot 45 + 2 \cdot 16 + 2 \cdot 2 = 81$

Careful about ambiguity! If code words were $a \to 0$, $b \to 1$, $c \to 01$ then 01 is ambiguous (01 = c? 01 = ab?)

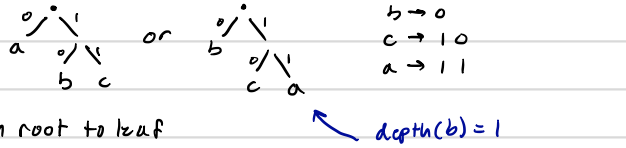
Need our encoding to be prefix-free: no code word is prefix of another code word

Use a full binary tree: each node has 0 or 2 children

left edge: label 0

right edge: label 1

decorate leaves w/ letters from Γ



encoding reads off edge labels on path from root to leaf

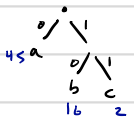
guaranteed to be prefix-free

depth(b) = 1

depth(c) = depth(a) = 2

So given encoding tree T, string S w/ letter counts f_l, $l \in \Gamma$, (think "frequency") size of encoded string is

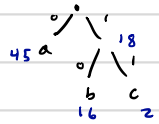
$$|S_e| = \sum_{l \in \Gamma} f_l \cdot \text{depth}(l)$$ ← want to minimize this wrt encoding tree T



$1 \cdot 45 + 2 \cdot 16 + 2 \cdot 2 = 81$

Clever re-write: lable internal nodes w/ sum of descendant counts

$\forall v \in T$, let cost(v) = sum of leaf node counts descending from v



$45 + 18 + 16 + 2 = 81$

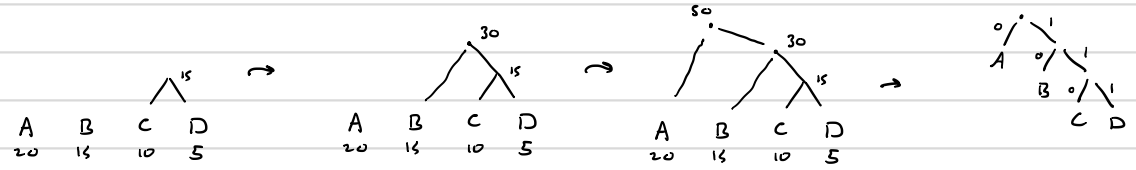
Then $|S_e| = \sum_{v \in T} \text{cost}(v)$

This suggests a greedy approach: Choose encoding of length inversely proportional to symbol frequency (i.e. most frequent symbols have smaller depth)

Idea behind algorithm: continually merge least frequent nodes until you have a binary tree

eg:

Sym.	freq
A	20
B	15
C	10
D	5



Symbol	code
A	0
B	10
C	110
D	111

total bits = 95

Since we aggregated up the tree @ each step, easy to check this problem exhibits optimal substructure (and the other 2 props of a greedy alg)

```
def Huffman(f)                                   ← array f[1...n] of frequencies
    Let T be an empty tree                       O(1)
    Let H be a priority queue, ordered by f      O(1)
    for i = 1 to n                               n · O(H insert time)
        insert(H, i)
    for K = n+1 to 2n-1                          (n-1)(2·O(H extract min) + O(1) + O(1) + O(H insert time))
        i = extract min(H)
        j = extract min(H)
        put node K in T with children i and j
        f[K] = f[i] + f[j]
        insert(H, K)
```

So complexity depends on implementation of the priority queue ← insert, min, extract, decrease

Heap-based: insert $O(1)$, extract min $O(\log n)$ ⇒ Huffman is $O(n \log n)$

van Emde Boas tree based: insert $O(\log\log n)$, extract min $O(\log\log n)$ ⇒ Huffman is $O(n \log\log n)$

Side note re: entropy

n symbols w/ freq $P_1, \dots, P_n$ $\quad \sum_i P_i = 1$

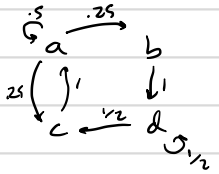
pull m total values according to the dist $(P_1, \dots, P_n)$

are number of bits to encode (using Huffman): $\sum_{i=1}^{n} m \cdot P_i \log\left(\frac{1}{P_i}\right)$ → single draw $H(P_1, \dots, P_n) := \sum_{i=1}^{n} P_i \log\left(\frac{1}{P_i}\right)$ ← metric/Shannon's entropy

Only applicable when iid! (one @ a time, indep, all pulled according to same dist).

Relatedly, Huffman only optimal in this situation → w/ restriction of 1 symbol → 1 code word

Say



aabdcabdd....

entropy $\approx \sum P_i \log(1/P_i)$

Matroids, processor scheduling
Set cover
DP
LP
NP

Matroids

Theory generalizing many greedy algorithms

Def'n a matroid $M=(S,\mathcal{I})$ is an ordered pair s.t.

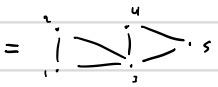
1. S is a finite set
2. $\mathcal{I} \subseteq \mathcal{P}(S)$ set of subsets (call them indep sets) s.t.
   $B \in \mathcal{I}$ and $A \subseteq B$, then $A \in \mathcal{I}$ — all subsets of each set included
3. exchange property: $A \neq B \in \mathcal{I}$ s.t. $|A| < |B|$, then $\exists\, x \in B-A$ s.t. $A \cup \{x\} \in \mathcal{I}$

eg Graphic Matroid

given undirected graph $G=(V,E)$, $M_G=(S_G,\mathcal{I}_G)$ defined via

$S_G := E$ (set of edges)

$A (\subseteq E)$ is in $\mathcal{I}_G$ iff A is acyclic (i.e. if $G_A=(V,A)$ is a forest)



$S = \{(1,2),(1,3),(2,3),\dots,(4,5)\}$

$\mathcal{I}_G$ = set of all forests of G

$= \{\{(1,2)\}, \{(1,2),(1,3)\}, \{(1,2),(1,3),(3,4)\}, \dots, \}$

$(1,3)$ is an extension of $\{(1,2)\}$

Can verify this is a matroid

Def'n For $A \in \mathcal{I}$, say $x \in S$ is an extension of A if $A \cup \{x\} \in \mathcal{I}$

Def'n $A \in \mathcal{I}$ is maximal if it has no extension

Thm All maximal $A \in \mathcal{I}$ have the same size

Proof (by contradiction) Say $\exists\, B \in \mathcal{I}$ s.t. $|B| > |A|$. Then exchange property says $\exists\, x \in B-A$ s.t. $A \cup \{x\} \in \mathcal{I}$ ⊗ contradicts maximality of A. □

Def'n A weighted matroid is a matroid $M=(S,\mathcal{I})$ along w/ strictly positive $w: S \to \mathbb{R}_{>0}$

extend to sets via: $A \subseteq S \Rightarrow w(A) = \sum_{x \in A} w(x)$

eg In graph matroid, w gives edge weights

Observe: minimum spanning tree can be cast as a "maximum indep set" problem by setting $w(\text{edge}) = C - \text{orig\_edge\_weight(edge)}$ (some const larger than)

So if we could find an alg. that solves the "find a maximum indep set" for matroids, we would have another MST algorithm!

```
def Greedy(M=(S,I), w)
    A = ∅                                   let n = |S|
    S = Sort S into decreasing order by w   O(n log n)
    for each x ∈ S
        if A ∪ {x} ∈ I                      Say checking A ∪ {x} ∈ I is O(f(n))
            A = A ∪ {x}                     ↑ oracle
    return A
```

total time: $O(n \log n + n f(n))$

Can show this exhibits all 3 prop (iterative, safe, optimal substructure) and so returns an optimal/maximal A.

Task scheduling as a matroid problem

eg CPU scheduling programs to run

Setup: $S = \{a_1, \dots, a_n\}$ n unit-time tasks

$d_1, \dots, d_n$ deadlines by which tasks must be completed, $1 \le d_i \le n$

$w_1, \dots, w_n$ nonnegative penalties (weights) we incur if a_i is not finished by d_i deadline

Schedule: perm. of S specifying when to do tasks

Can transform any schedule into early first: $\{\dots, a_i, \dots, a_j, \dots\} \rightarrow \{\dots, a_j, \dots, a_i, \dots\}$ (late, early, still early, still late)

Canonical form: early before late, early ordered w/ monotonically increasing deadlines (swap early if not)

Defn a set A of tasks is indep. if ∃ a way to schedule them so that no task is late

Let $\mathcal{I}$ = set of all indep. tasks

Clearly $\mathcal{I}$ is hereditary, it possesses the exchange property: if $|B| > |A|$, $B, A \in \mathcal{I}$, let K be the time when last task in A finishes, let x be the first task in B that finishes after K, then $A \cup \{x\} \in \mathcal{I}$.

∴ $(S, \mathcal{I})$ is a matroid and so ∃ greedy alg. to solve it → $O(n \log n + n f(n)) = O(n^2)$

↖ checking $A \cup \{x\} \in I$ takes $O(n)$

Last couple of algs

Horn Formulas

(pos) literal, (neg) literal, or

Boolean Formulas: $x = T$ or F, $x \wedge \bar{y} \Rightarrow z \vee a$

and, implication

Special Kinds: Horn Formulas: a list of Boolean formulas of the form:

1. implications: $(x_1 \wedge \cdots \wedge x_n) \Rightarrow y$ all pos. literals, "ands" on LHS, single literal on the right
2. negative clauses: $(\bar{x}_1 \vee \cdots \vee \bar{x}_n)$ all neg. literals, "ors" only

Goal: Given a set of Horn formulas, find values of all literals that makes it evaluate to T (or report if you can't)

eg $(x \wedge y) \Rightarrow z$, $(\bar{x} \vee \bar{w})$,
T ✔ T

Say $w = F$, $x = F$, $y = F$, $z = F$

counter eg $x \Rightarrow y$, $\bar{x}$, $\Rightarrow y$
$F \Rightarrow T$ ✗ $x = F$ $y = T$

Idea: Start w/ all F, only set T if you need to, i.e. when an $\Rightarrow$ says you need to

Recall: $p \Rightarrow q \equiv \bar{p} \vee q$

p	q	p⇒q
T	T	T
T	F	F
F	T	T
F	F	T

→ means we need to set $q = T$ to get T

def Greedy Horn (set of Horn formulas)
Set all literals to F
While an $\Rightarrow$ is not satisfied
  Set RHS to T
if all negative clauses are T
  return assignment
else
  return "cannot be satisfied"

$O(n^2)$ but can be reduced to $O(n)$

Correctness: Thm Any literal set to T by Greedy Horn must be set to true in any sat. assignment

Proof: Induction

eg $(x \wedge y) \Rightarrow z$, $(\bar{x} \vee \bar{w})$,

x	y	z	(x∧y)⇒z	(x̄∨w̄)
F	F	F	T	T

eg $\Rightarrow x$, $(\bar{x} \vee \bar{y})$, $x \Rightarrow y$

x	y	⇒x	(x̄∨ȳ)	x⇒y
F	F	F	T	T
T	F	T	T	F
T	T	T	F	T

∴ can't satisfy

Set Cover

Resource allocation, code testing, shift planning, etc

input: 1. Set $\mathcal{B}$

2. Subsets $S_1, \dots, S_n \subseteq \mathcal{B}$

output: a selection of subsets $I = \{S_{i_1}, \dots, S_{i_m}\}$ s.t. $\bigcup_{j=1}^{m} S_{i_j} = \mathcal{B}$ (i.e. the union of the selected subsets is $\mathcal{B}$)

Want to minimize $|I|$, the number of subsets selected

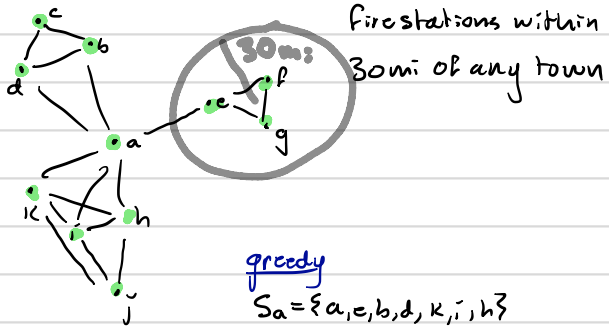
eg $\mathcal{B} = \{a, b, c, \dots, k\}$

$S_a = \{a, b, d, e, h, i, k\}$

$S_b = \{b, a, c, d\}$

$\vdots$

$S_k = \{k, a, h, i, j\}$



firestations within 30mi of any town

greedy heuristic: Choose the next subset based on the most # of uncovered cities

greedy

$S_a = \{a, e, b, d, k, i, h\}$

$S_f = \{f, g, e\}$

$S_c = \{c, b, d\}$

$S_j = \{j, k, i, h\}$

not optimal!

S_b, S_e, S_i

But! Not off by too much

Thm Say $|\mathcal{B}| = n$, optimal soln uses k sets, then greedy heur. uses $\leq k \ln(n)$ sets (called the approximation factor)

eg $k = 3$, $n = 11$, $3\ln(11) = 7.19$

Proof: Let n_t = # elements not covered by greedy @ iteration t

these remaining elems are covered by an optimal soln, so some optimal set has size $\geq n_t/k$ elements of $\mathcal{B}$ in it. $\therefore$

$$n_{t+1} \leq n_t - n_t/k = n_t(1 - 1/k)$$

(greedy picks set @ least this size)

repeat: $n_{t+1} \leq n_t(1-1/k) \leq n_{t-1}(1-1/k)^2 \leq \cdots \leq n_0(1-1/k)^t = n(1-1/k)^t \leq n(e^{-1/k})^t = ne^{-t/k}$

($n_t \leq n_{t-1}(1-1/k)$)

general fact via calculus: $1 - x \leq e^{-x}$, $\therefore 1 - 1/k \leq e^{-1/k}$

Now if $n_t < 1$, the program terminates, so want to solve $n_t < 1$ for t. Since $n_t < ne^{-t/k}$ if $ne^{-t/k} < 1$ So will $n_t < 1$. Solving: $ne^{-t/k} = 1 \Leftrightarrow e^{-t/k} = 1/n \Leftrightarrow -t/k = \ln(1/n) \Leftrightarrow t = -k\ln(1/n) = k\ln(n)$.

$\therefore$ will terminate within $t = k\ln(n)$ iterations. $\square$

Dynamic Programming

In greedy methods, making the greedy choice meant only one subproblem would remain.

If this is not the case (not 1 problem optimal substructure) will have multiple subproblems to analyze.

⇒ dynamic programming is designed for cases like this

hence: more generally applicable, but less efficient

Applies to optimization problems

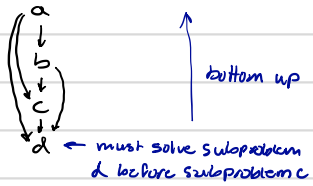
General Steps

Break the problem into smaller subproblems

tackle subproblems smaller first (bottom up typically)

use information from smaller problems to solve bigger ones

This is conceptualized with a **subproblem graph**: a DAG where vertices represent subproblems, edges $x \to y$ if determining an optimal solution for x directly involves considering an optimal solution for y



Note: typically, time to solve a subproblem is proportional to # of outgoing edges

subproblems = # vertices

So ∴ running time linear in # vertices + # edges

eg Longest incr. subsequence

Set of #'s $a_1, \dots, a_n$, find a subsequence $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ w/ $i_1 < i_2 < \dots < i_k$ w/ largest length K

5 2 8 6 3 6 9 7

Subproblem DAG:

← to compute the longest subseq ending w/ 7, need to compute longest subseq ending in any of its possible predecessors etc

Solve from the bottom up:

For v in reverse topological sort of (V, E) →

$L(v) = 1 + \max\{L(i) : (v,i) \in E\}$

(count vertices, not just edges)

Return $\max_v L(v)$

$L(5) = 1$ (no $(5,i) \in E$)

$L(2) = 1$

$L(8) = 2$

⋮

Designing a DP

look for

1. optimal substructure
2. overlapping subproblems

1. opt. substruct: optimal soln to problem contains optimal solns to subproblems → lets you draw a subproblem graph

eg unweighted shortest path $G=(V,E)$

Simple path (o.w. remove cycles)

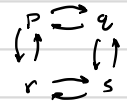
opt. substruct. since

$\text{Shortest}(u,v) = u \to \cdots \to w \to \cdots \to v$ (must be shortest(w,v) since o.w. could insert shortest(w,v) here and would contradict shortest(u,v))

counter eg unweighted longest path

$\text{longest}(p,r) = p \to q \to s \to r$

not $\text{longest}(s,r) = s \to q \to p \to r$



∴ no opt. substruct

2. Overlapping Subproblems: when a recursive alg encounters the same subproblems over & over

eg subproblem graph

Subproblem D encountered when solving A, B, & C

This leads to superiority of DP over recursive → can reduce exp. recursive time to polynomial time DP

Application Edit distance

DNA $S^1 = ACGTA$, $S^2 = ATCTG$ (note $|S^1| \neq |S^2|$ ncc.)

$d_E(S^1,S^2)$ = min # insertions, deletions, & substitutions needed to transform S^1 to S^2

$d_E(S^1,S^2) = 3$ convenient to write

$$\begin{matrix} A-CGTA \\ ATC-TG \end{matrix}$$ call the optimal one an alignment — cost 3

$$\begin{matrix} AC--GTA \\ ATCTG-- \end{matrix}$$ cost 5 (not minimal)

Applications in phylogeny, finding related genomes, extensions to other indels (Mingfu Shao)

Let's look for optimal substruct & overlapping subproblems

1. opt. substruct: Say strings $X = x_1 \cdots x_m$, $y = y_1 \cdots y_n$ (eg $S^1 = X = ACGTA$, $x_1 = A$, $x_2 = C$)

look @ prefixes (bottom up): $x_1 \cdots x_i$, $y_1 \cdots y_j$ $i \le m$, $j \le n$ → Call this subproblem $P(i,j)$

If solved $P(i,j)$, look @ rightmost column of alignment:

① $\begin{matrix} x_i \\ - \end{matrix}$ or ② $\begin{matrix} - \\ y_j \end{matrix}$ or ③ $\begin{matrix} x_i \\ y_j \end{matrix}$

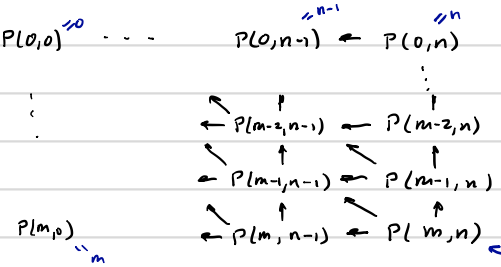
case ①: Contributes 1 to cost plus whatever the cost of aligning $x_1 \cdots x_{i-1}$ and $y_1 \cdots y_j$ → this is subproblem $P(i-1,j)$!

case ②: " " " " " " $x_1 \cdots x_i$ and $y_1 \cdots y_{j-1}$ → this is subproblem $P(i,j-1)$!

case ③: if $x_i = y_j$ then contrib. 0 plus cost of aligning $x_1 \cdots x_{i-1}$, $y_1 \cdots y_{j-1}$ → $P(i-1,j-1)$

if $x_i \neq y_j$ " " 1 " " " " → $P(i-1,j-1)$

thus, subproblem graph



So ∃ opt substruct, & clearly overlapping

P(m-1,n-1) used in problems P(m-1,n), P(m,n), & P(m,n-1).

in general, for $P(i,j)$, need only pick which of 3 cases is least cost:

$$P(i,j) = \min\left(1 + P(i-1,j),\ 1 + P(i,j-1),\ d_i[x_i,y_j] + P(i-1,j-1)\right)$$

$d_i[x_i,y_j] = \begin{cases} 0 & x_i = y_j \\ 1 & x_i \neq y_j \end{cases}$

∴

```
def EditDist(X,Y):
    for i=0,...,m
        P(i,0) = i
    for j=0,...,n
        P(0,j) = j                                   O(mn)
    for i=0,...,m        <- order doesn't matter
      for j=0,...,n
        P(i,j) = min( 1+P(i-1,j), 1+P(i,j-1), d_i[x_i,y_j]+P(i-1,j-1) )
    return P(m,n)
```

get actual alignment by keeping track of where min came from ↑ backtracking from $P(m,n)$

eg

		A	T	C	T	G
	0	1	2	3	4	5
A	1	0	1	2	3	4
C	2	1	1	1	2	3
G	3	2	2	2	2	3
T	4	3	2	3	2	3
A	5	4	3	3	3	3

∴ $P(m,n) = d_E(ACGTA, ATCTG) = 3$

↖ = match

↑ = blank in top of alignment

← = blank in bottom " "

```
A C G T A        or    A T C - T A
A T C T G              A - C G T G
cost 3                 cost 3
```

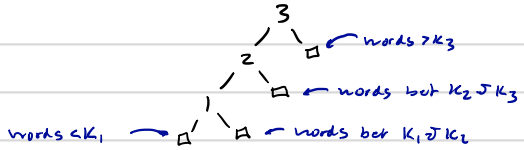
weighting ↖ or ↑ or ← by different amounts leads to alternative algs: Smith-Waterman, Needleman-Wunsch

example 2: instead of chain matrix mult (see book), problem in similar spirit: optimal binary search

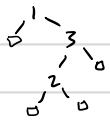
Translate French book to English, organize dict. for efficiency: word freq is Zipfian — take word freq into account

Some words may have no translation

Say only 3 words w/ keys $k_1 < k_2 < k_3$ (say lexicographic order) w/ freq/prob p, q, r



cost $3p + 2q + r$



cost $p + 3q + 2r$

etc

Idea is most frequent words should be highest in tree: cost = total tree traversal length over whol corpus

$\exists \binom{2n}{n}/(n+1) \times 4^n/(\sqrt{\pi}\, n^{3/2})$ binary trees on n nodes ⇝ can't brute force check for lowest cost tree

Also, absent word freqs will play a role (the □ in the above).